Preparing VerbTagGr for Modernization: Data Migration and Structural Validation of a Modern Greek Verb-Analysis Resource

Christonakis M., Lembessi P., Symeonaki E., Papakitsos E*

Department of Industrial Design and Production Engineering, University of West Attica, Athens, Greece

***Corresponding Author**

DOI: https://doi.org/10.68050/JAMS.2026.882

Received: 16 October 2026; Accepted: 22 October 2026; Published: 30 October 2026

ABSTRACT

This paper examines the data-preparation stage in the modernization of VerbTagGr, a computational tool designed to recognize the grammatical properties of one-word verb forms in Modern Greek. The project addresses a foundational rather than algorithmic problem: how to make an established linguistic resource usable in a contemporary software environment without changing its linguistic content. VerbTagGr originated in research on statistical morphosyntactic recognition and lemmatization and was initially implemented with Java, JavaServer Pages (JSP), and MySQL. Its data resources were subsequently extended in VerbTagGr++ to cover a broader range of verb forms and to support future lemmatization. The present work focuses on the VerbDB database, especially the verb_info and ms_codeexplain tables, and documents a workflow involving MySQL configuration, database creation, schema inspection, data import, record-count verification, CSV export, and conversion to Excel workbooks for later use in Python. The equalsmscodes table was retained for structural compatibility, although it contained no records in the version examined. The resulting workbooks provide a more accessible and portable representation of the source data and establish a practical bridge between the legacy database and a planned Python implementation. The thesis reports successful completion of the conversion process and describes checks intended to preserve character encoding, field correspondence, record completeness, and the mapping between verb endings and grammatical codes. However, it does not report a benchmark of recognition accuracy or a completed Python recognizer; these remain tasks for subsequent development. The contribution is therefore best understood as a reproducible data-engineering stage that reduces technical barriers to continued research while preserving the distinction between preparing linguistic resources and validating the behavior of a complete language-processing system.

Keywords: VerbTagGr; Modern Greek; verb lemmatization; database migration; linguistic data engineering.

1. INTRODUCTION

Computational analysis of natural language depends not only on algorithms but also on the quality, consistency, and accessibility of the linguistic resources those algorithms use. This is particularly apparent in morphologically rich languages, where a single lexical item may appear in numerous inflected forms and where surface forms do not always map transparently to a single stem. Modern Greek verbal morphology presents a demanding case: verb forms encode grammatical distinctions such as person, number, tense, voice, and mood, while stem alternations and irregular forms complicate straightforward recognition. A computational resource intended to identify these properties must therefore combine linguistic analysis with a carefully structured representation of forms, endings, and grammatical labels (Christonakis, 2026).

VerbTagGr was developed to address the recognition of one-word Modern Greek verb forms through a statistical approach grounded in extensive linguistic data. The source thesis situates this work in relation to the research of Lembessi (2005), which examined the statistical morphosyntactic recognition and lemmatization of the Modern Greek verb system. According to the account provided in the source, that research analysed

approximately 520,000 verb forms, identified 151,527 unique final graphemic sequences, and described 385 inflectional models, alongside exceptions and special forms. These figures indicate both the scale of the linguistic problem and the importance of organizing the underlying data systematically.

The original VerbTagGr implementation combined a MySQL database with a Java web service and a JSP user interface. Its early coverage focused on forms with unstressed endings in -a and -n, reflecting the availability of the corresponding data groups at that stage. A later extension, VerbTagGr++, expanded coverage to all available verb forms and introduced database structures relevant to lemmatization. The historical development is described in the source thesis with reference to Kroustalli (2005) and Zerva et al. (2016). Together, these stages produced a linguistic resource whose continued usefulness depends on its ability to be maintained, inspected, and integrated with newer software.

The present paper concerns a specific and bounded stage of that modernization. It does not claim to implement a new morphological recognition algorithm or to complete a Python version of the application. Instead, it examines the preparation of the existing VerbDB data for use by a future Python implementation. The practical work reconstructs and inspects the MySQL database, imports the relevant source data, checks the database schema and record counts, exports selected tables to comma-separated values (CSV), and converts those exports into Excel workbooks. The principal outputs are ms_codeexplain.xlsx and verb_info.xlsx. This transformation is intended to make the resource more accessible to developers and researchers while maintaining the meaning and structure of its linguistic records.

This distinction matters methodologically. A file-format conversion is not, by itself, evidence that a recognizer is accurate, that every linguistic ambiguity has been resolved, or that a future program will behave correctly. It is an enabling step: it makes data available in a form that can be inspected and consumed by a different programming environment. The value of this stage lies in reducing technical friction, documenting the data pathway, and preserving the relationships on which later recognition and lemmatization depend. The source thesis (Christonakis, 2026) reports that the work was completed successfully and that the resulting files are intended for the next phase of VerbTagGr development in Python.

The paper has four aims. First, it outlines the linguistic and technical background necessary to understand the role of VerbTagGr and VerbDB. Second, it describes the data-migration workflow as a methodological sequence rather than as a collection of software screenshots. Third, it analyses the outputs and the extent to which the reported checks support claims about data readiness. Fourth, it identifies limitations and proposes a set of validation steps for future work, including programmatic integrity checks and evaluation of the completed recognizer.

2. METHODOLOGY

2.1. Research design and scope

The work is a technical case study of preparing an existing linguistic database for reuse in a modern programming environment. Its primary object is the VerbDB database associated with VerbTagGr. The method is descriptive and procedural: it reconstructs the stages previously reported, identifies the role of each stage, and evaluates what the reported checks can reasonably establish. It is not an experimental comparison of competing recognition algorithms. No new corpus annotation, statistical model training, or quantitative accuracy evaluation is reported, and this paper does not infer such results.

The scope is restricted to the preparation and transformation of data. The original linguistic analysis and the grammatical code system are treated as inherited resources. The stated intention is to preserve their scientific content while changing the technical form in which selected data are stored and accessed. Consequently, the paper distinguishes three kinds of outcome: (1) database reconstruction and structural inspection; (2) export and conversion of the data; and (3) readiness for later software integration. Only the first two are directly completed. The third is an intended downstream benefit that requires separate testing in Python.

2.2. Linguistic and database resources

The source thesis (Lembessi, 2005) describes Modern Greek verb forms as combinations of a lexical or stem component and a grammatical component. The latter encodes features such as person, number, tense, voice, and mood. Aspectual distinctions and stem alternations can also be relevant to interpreting a form. Examples in the thesis include the alternation among forms related to βλέπω, είδα, and ιδ-, illustrating why a verb cannot always be analysed by simply removing a uniform suffix. In a statistical ending-based approach, the final graphemic sequence is a useful signal, but its interpretation depends on a consistent mapping between that sequence and grammatical information.

The underlying research by Lembessi (2005) is described as having identified a large inventory of final graphemic sequences and 385 inflectional models. The original VerbTagGr and its later VerbTagGr++ extension represent successive stages in making these resources operational. The database central to the present work contains three tables of particular relevance. The verb_info table stores final graphemic sequences, associated grammatical codes (ms_code), and additional indicators relevant to recognition and prospective lemmatization. The ms_codeexplain table maps these codes to human-readable grammatical attributes, including person-related information, tense, voice, mood, and form type. The equalsmscodes table is intended to store relationships among equivalent grammatical codes; the source thesis reports that it was empty in the version used but was retained for compatibility with the existing structure.

2.3. Tools and environment

The reported implementation (Christonakis, 2026) used MySQL Community Server to create and manage the database, MySQL Workbench to execute SQL statements and inspect table structures, and Microsoft Excel to review and convert exported records. The work was performed in a Windows environment. The MySQL configuration was adjusted to permit local file import and export operations, including LOAD DATA INFILE and SELECT ... INTO OUTFILE, subject to the server's file-access restrictions. The configuration file identified in the source was my.ini, located under the MySQL Server 8.0 installation directory in ProgramData.

The file-conversion workflow used CSV as an intermediate format. CSV is a straightforward exchange format, but its interpretation depends on consistent delimiters, line endings, quoting, and character encoding. These issues are especially important when a dataset contains Greek characters and diacritics. The source thesis therefore emphasizes UTF-8 encoding and visual inspection of the exported material. The final Excel files provide a tabular representation that can be inspected manually and read by Python libraries in a later implementation. Excel was used here as a delivery and inspection format, not as a replacement for a database system in every production context.

2.4. Procedure

The first stage was to install and configure MySQL Community Server and MySQL Workbench and establish a local connection. The MySQL80 Windows service was checked, and the server configuration was reviewed so that file-based import and export could be carried out. The source thesis describes modifying the secure-file-priv setting, which restricts the directories from which MySQL can import or to which it can export files. Because such settings have security implications, the operation is best understood as an environment-specific configuration step rather than as a universal recommendation to disable file restrictions. In a production setting, a narrowly scoped permitted directory and appropriately restricted file permissions would be preferable.

The second stage was the creation of VerbDB. The source thesis reports running the SQL script created.sql to establish the database and its tables, followed by data scripts named data_A.sql, data_AN.sql, and data_restN.sql. The successful creation of the database was checked using SHOW DATABASES. The database was then selected, and SHOW TABLES was used to verify the presence of verb_info, ms_codeexplain, and equalsmscodes. These operations established that the expected database objects were available for further inspection.

The third stage examined the schema. The DESCRIBE command was used for ms_codeexplain and, in a similar way, for the other tables. The inspection confirmed the presence of fields corresponding to the grammatical code and its explanatory features. In particular, the ms_code field provides the key association between the records describing endings and the table explaining their grammatical interpretation. The equalsmscodes table was also checked and found to contain no records in the version examined. Its retention was motivated by compatibility with the previous structure rather than by an immediate role in the reported workflow.

The fourth stage imported the linguistic data from CSV files using LOAD DATA INFILE. Attention was paid to character encoding, the correspondence between source fields and destination columns, and the number of rows imported. The source thesis reports the use of SELECT COUNT(*) to check record totals after import. These checks are useful for detecting missing or incomplete bulk imports. They do not alone prove that each individual value is correct, so they should be interpreted as one layer of verification rather than as exhaustive validation.

The fifth stage exported selected table contents to CSV files using SELECT ... INTO OUTFILE. For ms_codeexplain, the reported export selected fields including primaryKey, ms_code, person_partcplInfo, tense, voice, mood, and type. The verb_info table was exported in a corresponding way. No export file was produced for equalsmscodes because the table had no records. The CSV files then served as intermediate artefacts between MySQL and Excel.

The sixth stage converted the CSV files into Excel workbooks. The process included checking that characters displayed correctly, reviewing column organization, cleaning values where needed, removing unnecessary rows, and checking that the resulting tables remained complete and consistent. The principal outputs were ms_codeexplain.xlsx and verb_info.xlsx. The appendix to the source thesis also describes Greek-language worksheets created in the workbooks by replacing selected Latin or symbolic values with Greek equivalents while preserving the original structure and codes. This provides a more readable view for users familiar with Greek grammatical terminology. The existence of translated or localized worksheets should not, however, be confused with changes to the underlying linguistic analysis.

2.5. Verification logic

The reported verification can be understood as a sequence of checks at different levels. Database-level checks confirmed that the expected database and tables existed. Schema-level checks confirmed that the tables exposed the expected fields. Import-level checks compared record counts and attended to UTF-8 character encoding. Export-level checks produced CSV representations of the selected tables. Workbook-level checks involved inspection of character rendering, column layout, completeness, and consistency.

A stronger modern pipeline would formalize these checks as executable assertions. For example, the number of rows in a source table could be compared with the number of parsed CSV records and the number of worksheet data rows; the uniqueness and non-null status of key fields could be tested; and every ms_code used in verb_info could be checked against the set of codes represented in ms_codeexplain. The source thesis emphasizes consistency between these tables, but does not report a complete machine-generated audit with detailed pass/fail results for every such test. This paper therefore treats these additional checks as recommendations, not as completed outcomes.

2.6. Reproducibility and analytical approach

The source thesis documents the workflow with screenshots and an appendix describing the technical operations. This documentation supports procedural reproducibility by identifying the software, commands, table names, and principal files involved. Nevertheless, screenshots alone are less robust than a version-controlled set of SQL scripts, conversion scripts, and automated tests. A reproducible release should include a manifest of input files, tool versions, character encodings, export settings, record counts, and cryptographic checksums. If the original database and output workbooks are preserved, checksums can also help determine whether a later transformation has altered the files unexpectedly.

For the purposes of this paper, results are assessed in relation to the declared scope: whether the data were reconstructed, inspected, exported, converted, and prepared for downstream use. The paper does not attribute performance improvements to Excel or Python without measured evidence. Nor does it treat the intended future lemmatization capability as already delivered by the data-conversion stage.

3. RESULTS AND DISCUSSION

3.1. Reconstructed database structure

The first reported result is the successful reconstruction of the VerbDB database and the availability of its three expected tables: verb_info, ms_codeexplain, and equalsmscodes. The scripts used to create and populate the database provided the bridge from the existing SQL-based resource to a working local instance. The SHOW DATABASES and SHOW TABLES commands provided simple, direct confirmation that the database and tables were present. DESCRIBE inspections further established the field structure needed to interpret the stored information.

The two principal tables play complementary roles. The verb_info table connects final graphemic sequences with grammatical codes and supplementary indicators. The ms_codeexplain table gives those codes their linguistic interpretation by associating them with grammatical attributes. The ms_code field is therefore not merely an arbitrary identifier: it functions as a link between a surface-form pattern and a structured description of its grammatical properties. Preserving the integrity of this link is essential to the intended recognition process. If a code in verb_info were absent from ms_codeexplain, the system could identify a pattern but fail to retrieve a complete grammatical description. Conversely, inconsistent or duplicated code definitions could lead to unclear or conflicting outputs.

The third table, equalsmscodes, was found to contain no records in the examined version. Its absence of content does not mean that the database reconstruction failed; the table was part of the inherited schema and was retained for compatibility. However, the empty state should be explicitly documented in a future release so that downstream developers do not mistake an empty table for an import omission. If a future implementation begins using equivalent-code mappings, its semantics and expected contents will need to be defined and tested.

3.2. Migration to CSV and Excel

The second principal result is the production of the ms_codeexplain.xlsx and verb_info.xlsx workbooks. Their creation followed the export of the relevant MySQL tables to CSV and subsequent inspection and conversion in Excel. The resulting format is comparatively accessible to both technical and non-technical users. Researchers can review the rows and columns directly, while a Python application can load the tabular data using standard data-processing libraries. The files can also serve as intermediate artefacts during development, making it easier to inspect examples and compare the source records with the data consumed by a new implementation.

The use of CSV as an intermediate representation is practical because it separates database extraction from spreadsheet formatting. It also makes the migration steps more transparent: the contents can be examined as text before they are loaded into a workbook. Nevertheless, CSV is not a fully self-describing data format. It does not inherently preserve database types, primary-key constraints, foreign-key relationships, indexes, or all details of null values. Depending on the export and import settings, delimiters, quotes, line endings, and encoding can affect the interpretation of a field. Consequently, the conversion must be treated as a controlled transformation, not as a guarantee that every property of a relational database is preserved automatically.

The source thesis reports attention to UTF-8 encoding and to the correct display of Greek characters and accents. This is a significant practical requirement. Corrupted or normalized inconsistently, a Greek string may become unusable for matching or may be mistaken for a different form. For a resource whose purpose is to match graphemic sequences, text integrity is part of the data's linguistic integrity. The use of visual checks is therefore appropriate, although automated checks should complement them. A future pipeline could test a

known set of Greek strings with tonos and other relevant characters, compare round-tripped values exactly, and report any replacement characters or unexpected normalization.

3.3. Data quality and preservation of linguistic content

The thesis states that the conversion was completed without loss, alteration, or change to the scientific content of the data. The described workflow provides a basis for this conclusion: source tables were inspected, imported records were counted, and output workbooks were reviewed. However, the strength of a data-preservation claim depends on the precision of the comparison performed. Matching row counts can establish that the number of records is consistent, but it cannot establish that every value in every row is unchanged. Visual inspection can detect conspicuous problems, but it is not exhaustive for a large table.

For a rigorous audit, the next version of the pipeline should compare source and output records systematically. At minimum, this would include row counts, column names and order, null-value counts, unique-key counts, and exact comparisons of selected or all values after accounting for expected representation changes. For each row, the set of codes in verb_info should be checked against the set of codes in ms_codeexplain. Where primary keys are intended to be unique, duplicate-key tests should be applied. Any changes made for readability—such as translating selected field values into Greek on separate worksheets—should be recorded as a presentation layer and kept distinct from the canonical values consumed by code.

This distinction is particularly important because the resource embodies linguistic analysis developed through earlier research. The current project aims to preserve that analysis, not revise it. A technical migration should therefore avoid silent correction of grammatical labels, changes to codes, or the removal of forms that appear unusual. Potential linguistic anomalies should be recorded for review by a qualified researcher rather than automatically changed during conversion. Separating the original dataset, the transformed data, and any human-readable localized view supports both provenance and accountability.

3.4. Contribution to modernization of VerbTagGr

The contribution of the project lies in making a legacy resource easier to use in a new technical context. The original application architecture combined MySQL with a Java web service and JSP interface. The planned next stage uses Python to rebuild or extend the recognition functionality. The work described here does not itself perform that rewrite, but it prepares a form of the data that can be consumed during implementation. In software projects, this intermediate layer is often decisive: a new algorithm cannot be reliably developed until its input resources are accessible and their structure is understood.

The two workbooks provide a handoff point between database maintenance and algorithm development. They make the main mappings inspectable, enable developers to prototype loading and lookup routines, and reduce the need to reproduce the legacy database environment for every exploratory task. They may also facilitate collaborative work, since linguistic specialists can review table contents without needing to interact directly with a MySQL server. In this sense, the work performs a data-engineering function within a broader language-technology project.

3.5. Implications for recognition and lemmatization

VerbTagGr's intended function depends on matching a verb form or its ending to a code and then interpreting that code through a structured table. In the workflow described in the thesis, the verb_info table is consulted to retrieve the relevant ms_code, and ms_codeexplain provides the grammatical attributes associated with that code. This separation between recognition evidence and explanatory metadata is conceptually useful. It permits a compact lookup structure while keeping the grammatical descriptions centralized.

The wider VerbTagGr++ resource is also described as containing structures relevant to lemmatization. Lemmatization maps an inflected form to a canonical lexical form, and Greek verb forms can require more than a mechanical suffix-stripping operation because of stem alternations and irregular paradigms. The availability of lemmatization-related data is therefore an important basis for future work, but data availability

should not be conflated with a completed lemmatization function. The source thesis explicitly frames the Python implementation and completion of the tool as subsequent work. Accordingly, the present results support the feasibility of continuing development, not a claim that end-to-end lemmatization has already been implemented or evaluated.

A complete successor system will need to define how it handles ambiguous forms, multiple possible analyses, irregular forms, accentuation, normalization, and input that is not represented in the database. It will also need to decide whether the recognition unit is always a complete one-word verb form or whether the interface will eventually support multiword and periphrastic constructions. The source thesis notes that some Greek tenses are expressed periphrastically, for example with forms equivalent to “have written” or “had read.” Since the current resource is described primarily in terms of one-word forms, support for periphrastic constructions should be treated as a separate design question rather than assumed to follow automatically from the database migration.

3.6. Limitations of the reported evidence

The source thesis provides a procedural account and screenshots of key steps, but it does not report a quantitative evaluation of the transformed files or a performance test of a Python implementation. No benchmark corpus, precision, recall, F-score, processing time, or comparison with another morphological analyser is presented in the described work. This is consistent with the project’s bounded scope, but it limits the conclusions that can be drawn. The strongest supported claim is that the source database was reconstructed and the selected tables were converted into Excel files through a documented process. Claims about recognition accuracy, coverage, speed, or practical improvements in the user experience require subsequent evidence.

There is also a distinction between successful conversion and reproducibility at scale. The use of screenshots supports human understanding, yet screenshots cannot fully replace executable scripts and test outputs. Manual spreadsheet editing can introduce accidental changes or make the transformation difficult to repeat exactly. A robust next iteration should automate conversion wherever possible, preserve a read-only copy of the original data, log every transformation, and run the same validation suite each time the data are regenerated.

The database configuration also deserves a security qualification. Changing secure-file-priv may be necessary in a controlled local environment to enable export operations, but broad relaxation of file-access restrictions can increase risk. A documented migration should specify the permitted directory and restore appropriate restrictions after the operation. This is not evidence that the reported project was insecure; rather, it is a general operational consideration for making the procedure safe and repeatable on other machines.

3.7. Recommendations for the next stage

Several steps would strengthen the transition from the prepared workbooks to a working Python tool. First, a canonical schema should be documented, including the meaning and expected type of each column, key constraints, permitted values, encoding assumptions, and the relationship between the two principal tables. Second, the conversion should be automated and version-controlled. A Python script could read the CSV files, validate expected columns and types, check key relationships, and write the workbook or another target format. Third, integrity tests should compare counts and values across the source database, CSV exports, and final workbooks. The checks should produce a concise report that can be retained with each release. Fourth, the new recognizer should be evaluated against a manually verified test set containing regular forms, irregular forms, forms with stem alternations, ambiguous endings, accented Greek characters, and forms not present in the resource. Evaluation should distinguish coverage from correctness: a tool may return analyses for many inputs but still assign incorrect grammatical features. Precision and recall, together with error categories, would make it possible to identify systematic weaknesses. Fifth, lemmatization should be evaluated separately from grammatical recognition, because correctly assigning features does not guarantee that the canonical lemma is recovered correctly.

Finally, a release package should preserve the original data, the transformed canonical data, localized presentation sheets if useful, conversion scripts, validation reports, and documentation. Such a package would help future maintainers understand which elements are inherited from the linguistic research, which are transformed for technical reasons, and which are introduced by later software. This provenance-oriented approach would support the project's stated aim of preserving linguistic content while modernizing the implementation.

3.8. Data representation and interoperability

The transition from a relational database to spreadsheets raises a broader question about how linguistic resources should be represented across the life cycle of a software project. A relational database is designed to enforce structured relationships and can express constraints that are not automatically carried over into a spreadsheet. A workbook, by contrast, is convenient for inspection, annotation, and exchange, but allows users to edit cells in ways that may violate the assumptions of the original schema. The conversion should therefore be treated as a change of representation, not a migration that makes all database properties equivalent. The schema and constraints that matter to the application need to be documented independently of the file format.

For VerbTagGr, the most important relationship is the association between the graphemic sequence and the ms_code in verb_info, and the interpretation of that code in ms_codeexplain. The spreadsheet representation should preserve the exact values used as identifiers, including any leading zeros or symbols if these occur in the source data. Identifiers should not be automatically converted to numeric values, rounded, or reformatted by spreadsheet software. Similarly, empty cells, explicit nulls, and empty strings should be distinguished wherever the application semantics require that distinction. These are general data-management concerns rather than defects reported in the source thesis, but they are directly relevant to preserving the resource during later processing.

Interoperability also depends on establishing a clear contract between the data producer and the consuming application. The future Python code should not rely solely on column position or on the visual order of worksheet columns. It should identify columns by stable names, check that required columns are present, and stop with a clear diagnostic if the input file does not match the expected schema. This makes the integration more resilient to harmless presentation changes, such as rearranging columns for human readers. Where localized worksheets are included, the canonical machine-readable data should be clearly identified so that the application does not accidentally load translated labels in place of canonical values.

3.9. A layered model of validation

The reported workflow includes several appropriate basic controls, but validation can be strengthened by separating it into distinct layers. The first layer is structural validation: do the expected database and tables exist, and do their columns have the expected names and types? The second is completeness validation: do the row counts remain consistent through import and export, and are there unexpected empty records? The third is relational validation: can each code used in the recognition table be interpreted through the explanatory table? The fourth is textual validation: are Greek characters, accents, and punctuation preserved through the full round trip? The fifth is semantic validation: do the records express the intended linguistic analysis? These layers answer different questions and should not be treated as interchangeable.

Structural and completeness validation are relatively straightforward to automate. A test script can compare the expected set of table and column names with the observed schema and record counts. Relational validation can compare the distinct ms_code values found in verb_info with the codes defined in ms_codeexplain and list any unmatched values. Textual validation can compare source and destination strings exactly, while allowing only explicitly approved transformations such as a separate translated display column. Semantic validation is more demanding because it may require linguistic expertise and reference judgments. A technically consistent record can still contain a linguistically incorrect label inherited from the source; conversely, an unusual form should not be treated as an error merely because it is rare.

This layered approach supports more precise reporting. Rather than stating only that the data are "correct," a release could state which tests were performed, what they measured, and what exceptions were found. For example, it could report that table creation succeeded, that source and output row counts matched, that all codes resolved, and that a specified set of Greek strings passed round-trip checks. If a particular semantic audit has not been carried out, that limitation should remain explicit. Such reporting would make the dataset easier to maintain and would help future developers distinguish verified properties from assumptions.

The source thesis reports the use of SELECT COUNT(*) and inspection of the output tables. Those steps are meaningful and proportionate, focused on conversion. The proposed layered model does not diminish that work; it identifies a natural path from manual verification toward automated quality assurance as the tool matures. Once implemented, the same checks can be rerun after every update to the source data, reducing the chance that later changes introduce unnoticed inconsistencies.

3.10. Research significance and maintainability

A linguistic resource often outlives the software environment in which it was first deployed. The value of a data-preparation project can therefore be assessed not only by whether it produces a new file, but also by whether it lowers the cost of future maintenance. In the VerbTagGr case, the underlying linguistic resource is the result of substantial prior analysis, while the surrounding application technology has changed. Preparing the database for use in Python creates a route for continued access to the earlier work without requiring the new implementation to reproduce every detail of the old deployment environment.

The maintenance benefit is strongest when the transformation is transparent. A future researcher should be able to answer several questions: which files were used as inputs; which SQL scripts created and populated the database; which tables were exported; how the CSV fields were delimited and quoted; which encoding was used; what cleaning or formatting was performed in Excel; and which files are considered canonical for software use. The source thesis records many of these steps and includes a technical appendix. A next iteration could turn this descriptive record into a compact migration package containing the scripts, a README, a schema description, and a validation report.

Maintainability also depends on separating data from application logic. The existing code system and its explanations should be stored as data, while the Python implementation should provide the procedures that read and interpret them. This separation makes it possible to update the implementation without rewriting the linguistic resource, and to revise data only through a documented process. It also allows future contributors to inspect a disputed grammatical mapping without needing to understand the entire application architecture. For a resource intended to support research, this separation improves auditability and encourages collaboration between computational developers and linguists.

The project's approach is consequently relevant beyond this individual tool. Similar problems arise whenever legacy linguistic resources are migrated from a database, a collection of text files, or a proprietary format into a new environment. The main lesson is that migration should preserve provenance and meaning as carefully as it changes the technical representation. Portability is useful, but traceability is what allows later users to establish that a new implementation is still using the intended resource.

3.11. Threats to validity and boundaries of inference

Several boundaries should be kept in view when interpreting the reported outcomes. First, the paper relies on the description in the source thesis rather than on an independent re-run of every migration command against the original development environment. The existence of the resulting workbooks and the documented steps support the reported account, but the present analysis does not independently certify each record. Second, the source thesis reports visual checks and record-count verification, but does not supply a full audit log with row-level comparison results. The preservation claim is therefore supported procedurally but not quantified through a complete published integrity report.

Third, the linguistic data themselves inherit the decisions, scope, and limitations of the prior research. This project does not re-evaluate the 385 inflectional models or the classification of all listed endings. It also does not determine whether the resource covers every contemporary use of Modern Greek verbs. The reported coverage of VerbTagGr++ is a property attributed to the earlier extension in the source thesis, not a new coverage measurement made during the conversion stage. Fourth, the future Python implementation may introduce errors even if the workbooks are internally consistent. Loading, normalization, matching, code lookup, and presentation are separate operations, each of which requires testing.

These boundaries do not undermine the project's stated aim. They help locate the contribution accurately: a necessary preparatory stage was completed, and the data are presented in a more accessible form for further implementation. A careful account of what has and has not been demonstrated is especially important in future implementations, because it prevents technical feasibility from being mistaken for empirical performance. The next stage can build on the outputs while treating the validation and evaluation of the completed recognizer as distinct research tasks.

4. CONCLUSION

This paper has examined the data-preparation stage of the modernization of VerbTagGr, a resource for statistical morphosyntactic recognition of Modern Greek one-word verb forms. The work is grounded in an existing linguistic resource and its earlier implementations, and focuses on making the VerbDB data accessible for a planned Python-based continuation. The reported workflow includes configuring MySQL, creating and inspecting the database, importing data, checking record counts, exporting the principal tables to CSV, and converting them to Excel workbooks.

The principal outputs are ms_codeexplain.xlsx and verb_info.xlsx. The first provides the grammatical interpretation associated with analysis codes; the second connects final graphemic sequences and related information to those codes. The equalsmscodes table was retained as part of the schema despite containing no records in the version examined. Together, the outputs establish a practical handoff between the legacy database and future development. The work also emphasizes the importance of preserving Greek characters, maintaining consistent fields, and documenting the conversion process.

The contribution should be characterized precisely. It is a data-engineering and migration contribution, not a completed reimplementation of the recognition algorithm. The source thesis reports that the conversion process was completed successfully, but does not provide quantitative evidence of recognition accuracy or a completed Python system. Future work should therefore automate the pipeline, add machine-verifiable integrity tests, document the canonical schema, and evaluate recognition and lemmatization independently against a manually checked test set.

A reliable linguistic application requires more than a modern programming language: it also requires transparent, consistent, and traceable data. By preparing the VerbDB resources for renewed use while retaining their underlying linguistic organization, the project creates a foundation for the next stage of VerbTagGr. Its long-term value will depend on how carefully this foundation is tested, maintained, and integrated into the future application.

REFERENCES

1. Christonakis, M. (2026). Upgrading the VerbTagGr language tool with a modern PC programming language [in Greek]. Diploma thesis, University of West Attica, Department of Industrial Design and Production Engineering, Athens, Greece.
2. Holton, D., Mackridge, P., & Philippaki-Warburton, E. (1999). Grammar of the Greek language [in Greek]. Patakis.
3. Kroustalli, S. (2005). VERBTAGGR web-based tool for statistical morphosyntactic recognition of one-word Modern Greek verb forms [in Greek]. Diploma thesis, National and Kapodistrian University of Athens, Department of Informatics and Telecommunications, Athens, Greece.

4. Lembessi, P. (2005). Statistical morphosyntactic recognition and lemmatization of the Modern Greek verb system. Doctoral thesis, Université Marc Bloch, Strasbourg, France.
5. Lembessi, P. (2007). VerbTagGr: Web-based tool for statistical morphosyntactic recognition of one-word Modern Greek verb forms. Paper presented at the Poland L&T Conference.
6. McKinney, W. (2010). Data structures for statistical computing in Python. In S. van der Walt & J. Millman (Eds.), Proceedings of the 9th Python in Science Conference (pp. 51–56).
7. MySQL. (2024). MySQL 8.0 Reference Manual. https://dev.mysql.com/doc/refman/8.0/en/
8. Zerva, I., Koletsi, V., & Moragianni, N. (2016). VERBTAGGR++ web environment for statistical morphosyntactic recognition of one-word Modern Greek verb forms. Bachelor's thesis, National and Kapodistrian University of Athens, Department of Informatics and Telecommunications, Athens, Greece.
9. Klairis, C., & Babiniotis, G. (1999). Grammar of Modern Greek: Structural-functional and communicative [in Greek]. Ellinika Grammata.